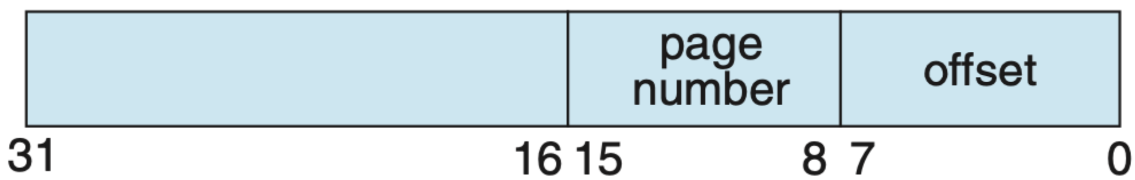
PROGRAMMING PROJECTS

Designing a Virtual Memory Manager

This project consists of writing a program that translates logical to physical address space of size 2^16 = 65,536 bytes. Your program will read from a file containing logical addresses and, using a TLB and a page table, will translate each logical address to its corresponding physical address, and output the value of the byte stored at the translated physical address.. Your learning goal is to use simulation to understand the steps involved in translating logical to physical addresses. This will include resolving page faults using demand paging, managing a TLB, and implementing a page-replacement algorithm.

Specific



Your program will read a file containing several 32-it integer numbers that represent logical addresses. However, you need only be concerned with 16-bit addresses, so you must mask the rightmost 16 bits of each logical address. These 16 bits are divided into (1) an 8-bit page number, and (2) an 8-bit offset. Hence, the addresses are structured as shown as:

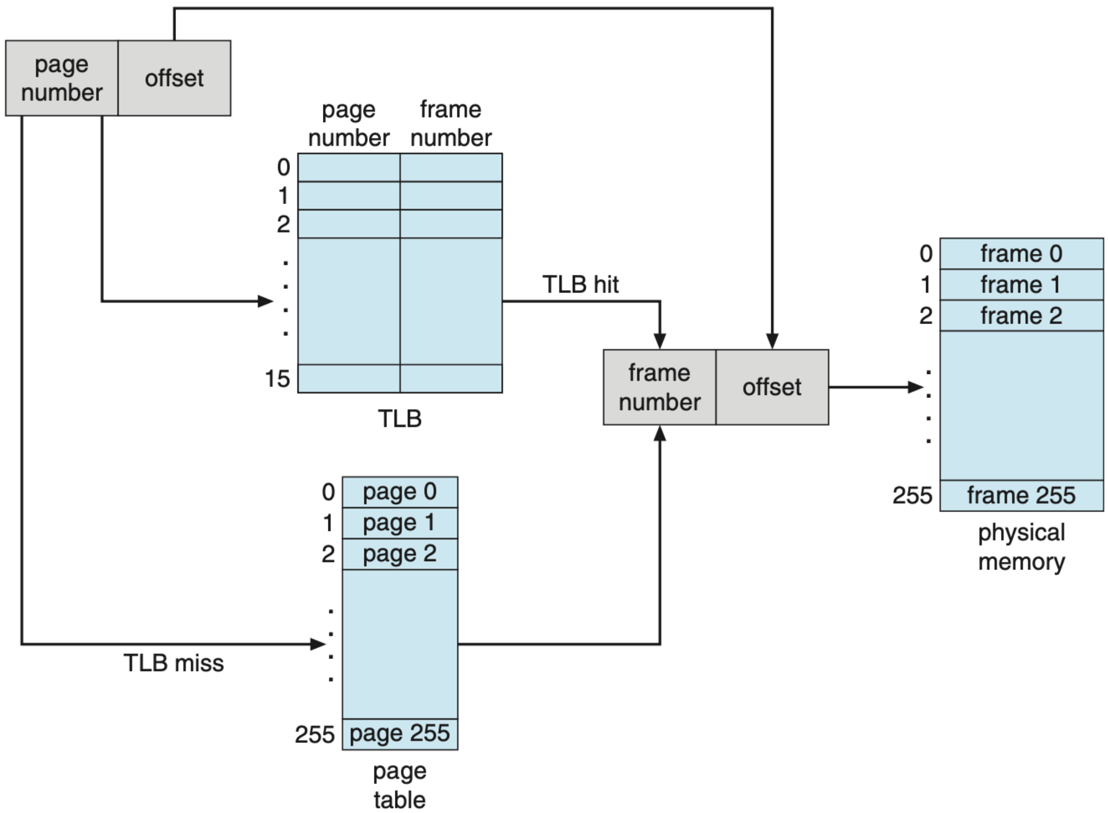
Other specifics include the following:

- **2^8 (256) entries in the page table**
- **Page size of 2^8 bytes (256 bytes per page)**
- **16 entries in the TLB**
- **Frame size of 2^8 bytes (256 bytes), so one page == one frame. This is not normally the case, and it won't be the case in the second part of this simulation.**
- **256 frames**
- **Physical memory of 65,536 bytes (256 frames x 256-byte frame size)**

Also, your program need only be concerned with reading logical addresses and translating them to their corresponding physical addresses. You don't need to support writing to the logical address space.

Address Translation

Your program will translate logical to physical addresses using a TLB and page table, as outlined in section 9.3. First, the page number is extracted from the logical address, and the TLB is consulted. In the case of a TLB hit, the frame number is obtained from the TLB. In the case of a TLB miss, the page table must be consulted. In the latter case, either the frame number is obtained from the page table, or a page fault occurs. A visual representation of the address-translation process is:



Handling Page Faults

Your program will implement demand paging as described in Section 10.2. The backing store is represented by the file BACKING_STORE.bin, a binary file of size 65, 536 bytes. When a page fault occurs, you will read in a 256-byte page from the BACKING_STORE and store it in an available page frame in physical memory. For example, if a logical address with page number 15 resulted in a page fault, your program would read page 15 from BACKING_STORE (remember that pages begin at 0 and are 256 bytes in size), and store it in a page frame in physical memory. Once this frame is stored (and the page table and TLB are updated), subsequent accesses to page 15 will be resolved by either the TLB or the page table.

You will need to treat **BACKING_STORE.bin** as a random-access file, so that you can randomly seek to certain positions of the file for reading. We suggest using the standard C library functions for performing I/O, including **fopen(), fread(), fseek(),** and **fclose()**.

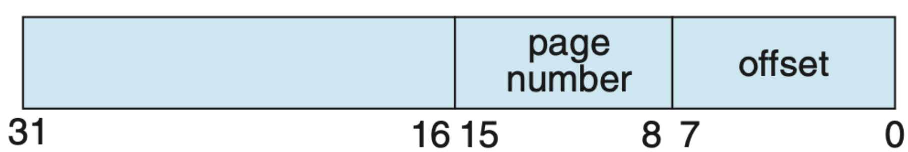
The size of physical memory is the same as the size of the virtual address space -- 65,536 bytes -- so you do not need to be concerned about page replacements during a page fault. Later, we describe a modification to this project using a smaller amount of physical memory; at that point, a page-replacement strategy will be required.

Test File

We provide the file **addresses.txt**, which contains integer values representing logical addresses ranging from 0 to 65,535 (the size of the virtual address space.) Your program will open this file, read each logical address, and translate it to its corresponding physical address, and output the value of the signed byte at the physical address.

How to Begin

First, write a simple program that extracts the page number and offset, based on: from the following integer numbers: **1, 256, 32768, 128, 65534, 33153 (hint: divide each by 256, and calculate the amount(page) and the remainder(offset) – example, 256/256 == (page) 1 with remainder (offset)0).**



Perhaps the easiest way to do this is by using the operators for big-masking and bit-shifting. Once you can correctly establish the page number and offset from an integer number, you are ready to begin.

In more detail…

Here is an example of bit masking: we want to mask off the page number to get the offset, how to do this? **Let's look at the long version first…**

Let `unsigned x` be the entire page number/offset, then

```
unsigned offset_mask = 0xFF;  // masks everything except bits 0-7
unsigned page_mask = 0xFF00;  // masks everything except bits 8-15

unsigned offset = x & offset_mask;       // just the offset
unsigned page_shifted = x & page_mask;   // just the page, but shifted
unsigned page = page_shifted >> 8;       // just the page
```

What's the short version ?

```
unsigned page(  unsigned x) { return ((x & 0xFF00) >> 8); }

unsigned offset(unsigned x) { return x & 0xFF; }
```

Initially, we suggest that you bypass the TLB, and use only a page table. You can integrate the TLB once your page table is working properly. Remember, address translation can work without a TLB; the TLB just makes it faster. When you are ready to implement the TLB, recall that it has only sixteen entries, so you will need to use a replacement strategy when you update a full TLB. You may use either a FIFO or an LRU policy for updating your TLB.

How to Run Your Program

Your program should run as follows: **./memmgr addresses.txt**

Your program will read in the file **addresses.txt**, which contains 1,000 logical addresses ranging from 0 to 65,535. Your program is to translate each logical address to a physical address, and determine the contents of the signed byte stored at the correct physical address. (Recall that in the C language, the char data type occupies a byte of storage, so we suggest using char values.)

Your program is to output the following values:

1. **The logical address being translated** (the integer value being read from addresses.txt)
2. **The corresponding physical address** (what your program translates the logical address to)
3. **The signed byte value stored in physical memory** at the translated physical address.

We also provide the file **correct.txt**, which contains the correct output values for the file **addresses.txt**. You should use this file to determine if your program is correctly translating logical to physical addresses.

Statistics

After completion, your program is to report the following statistics:

1. **Page-fault rate** -- the percentage of address references that resulted in page faults.
2. **TLB hit rate** -- the percentage of address references that were resolved in the TLB.

Since the logical address in addresses.txt were generated randomly, and do not reflect any memory access locality, do not expect to have a high TLB hit rate.

Page Replacement

Thus far, this project has assumed that physical memory is the same size as the virtual address space. In practice, physical memory is typically much smaller than a virtual address space. This phase of the project now assumes using a smaller physical address space, with 128 page frames rather than 256. This change will require modifying your program so that it keeps track of free page frames, as well as implementing a **page-replacement policy using either FIFO or LRU** (section 10.4) to resolve page faults when there is no free memory.

Resources

See this website for helpful background:
http://basen.oru.se/kurser/os/2019-2020-p4/labbar/labb-3/index.html

Files for this assignment can be found on Slack #project-help and on Titanium in file

memmgr_sample_code_input_files.zip

- memmgr.c
- addresses.txt
- correct.txt
- BACKING_STORE.bin

Example Output

```
NOT CORRECT -- ONLY READS FIRST 20 ENTRIES... TODO: MAKE IT READ ALL ENTRIES

logical: 16916 (page:  66, offset:  20) ---> physical:    20 -- passed
logical: 62493 (page: 244, offset:  29) ---> physical:   285 -- passed
logical: 30198 (page: 117, offset: 246) ---> physical:   758 -- passed
logical: 53683 (page: 209, offset: 179) ---> physical:   947 -- passed
logical: 40185 (page: 156, offset: 249) ---> physical:  1273 -- passed

logical: 28781 (page: 112, offset: 109) ---> physical:  1389 -- passed
logical: 24462 (page:  95, offset: 142) ---> physical:  1678 -- passed
```

```
logical: 48399 (page: 189, offset:  15) ---> physical:  1807 -- passed
logical: 64815 (page: 253, offset:  47) ---> physical:  2095 -- passed
logical: 18295 (page:  71, offset: 119) ---> physical:  2423 -- passed

logical: 12218 (page:  47, offset: 186) ---> physical:  2746 -- passed
logical: 22760 (page:  88, offset: 232) ---> physical:  3048 -- passed
logical: 57982 (page: 226, offset: 126) ---> physical:  3198 -- passed
logical: 27966 (page: 109, offset:  62) ---> physical:  3390 -- passed
logical: 54894 (page: 214, offset: 110) ---> physical:  3694 -- passed

logical: 38929 (page: 152, offset:  17) ---> physical:  3857 -- passed
logical: 32865 (page: 128, offset:  97) ---> physical:  4193 -- passed
logical: 64243 (page: 250, offset: 243) ---> physical:  4595 -- passed
logical:  2315 (page:   9, offset:  11) ---> physical:  4619 -- passed
logical: 64454 (page: 251, offset: 198) ---> physical:  5062 -- passed

ONLY READ FIRST 20 entries -- TODO: change to read all entries

ALL logical ---> physical assertions PASSED!
!!! This doesn't work passed entry 24 in correct.txt, because of a duplicate page table
entry
--- you have to implement the PTE and TLB part of this code

        ...done.
Program ended with exit code: 0
```

Submission

Turn in the code for this homework by uploading your project to a public repository on GitHub. While you may discuss this homework assignment with other students, you must complete the work on your own.

To complete your submission, print the following sheet, fill out the spaces below, and submit it to Titanium by the deadline. Failure to follow the instructions exactly will incur a **10%** penalty on the grade for this assignment.

CPSC 351 Project: Virtual Memory Manager, due 6 May 2021

Your name: Jacob Nguyen

Repository *(print)*: **https://github.com/** barrotbake/memory_manager

Verify each of the following items and place a checkmark in the correct column. Each item incorrectly marked will incur a 5% penalty on the grade for this assignment

Finished	Not finished	
☒	☐	Created functions that correctly calculate the offset and page of a given virtual address
☒	☐	Created a page table, that contains the frame of a given page, and which will page fault if the desired page is not in memory (this will happen: (A) when the program is first run and physical memory is empty, and (B) if only half as many physical frames as pages in the page table
☒	☐	Given a given logical address, checks the page table to find the corresponding physical address
☒	☐	Correctly reads the given physical address for the char value stored there
☒	☐	Goes to the BACKING_STORE and reads in the corresponding page into a free frame in physical memory. If there are only 128 frames, it must replace a frame to do this.
☒	☐	Implemented a Translation Lookaside Buffer (TLB) to store the most recently read-in page, AND checks the TLB first when decoding a logical address.
☒	☐	Do following when reading a logical address that is not in the TLB/Page table: Check TLB → (TLB miss) Check Page Table → (Page table miss) Page fault → read page from BACKING_ STORE → updates physical memory → updates Page table → updates TLB → reads value from physical memory..
☒	☐	Follows this flow diagram when has a TLB hit: Check TLB → Gets frame and offset → reads value from physical memory
☒	☐	Do following when has a TLB miss but a Page table hit → Check TLB → (TLB miss) → Checks Page table → Updates TLB → Gets frame and offset → reads value from physical memory
☒	☐	**Page-fault rate** -- the percentage of address references that resulted in page faults.
☒	☐	**TLB hit rate** -- the percentage of address references that were resolved in the TLB
☒	☐	Now modify your program so that it has only 128 page frames of physical memory (but still has 256 entries in the page table)
☒	☐	Program now keeps track of the free page frames, as well as implementing a **page-replacement policy using either FIFO or LRU**
☒	☐	Project directory pushed to new GitHub repository listed above

Fill out and print this page, and submit it on Titanium on the day this project is due.